Interrupt and Exception Handling
for Multi-Streaming Digital Processors

By Inventors
Mario D. Nemirovsky, Adolfo M. Nemirovsky, and Nerendra Sankar

Field of the Invention

The present invention is in the field of digital processors, and pertains more particularly to such devices capable of executing multiple processing streams concurrently, which are termed multi-streaming processors in the art.

Cross-Reference to Related Documents

The present application is a continuation-in-part (CIP) of prior co-pending patent applications 09/216,017, now U.S. Patent 6,477,562, 09/240,012, now U.S. Patent 6,292,888, and 09/273,810, now U.S. Patent 6,389,449, all three of which are incorporated herein in their entirety by reference.

06/05/00

Background of the Invention

Multi-streaming processors capable of processing multiple threads are known in the art, and have been the subject of considerable research and development. The present invention takes notice of the prior work in this field, and builds upon that work, bringing new and non-obvious improvements in apparatus and methods to the art. The inventors have provided with this patent application an Information Disclosure Statement listing a number of published papers in the technical field of multi-

Fig. 2A is a flow chart illustrating one method whereby a thread in one stream forks a thread in another stream and later joins it.

Fig. 2B is a flow chart illustrating another method whereby a thread in one stream forks a thread in another stream and later joins it.

Fig. 3 is a flow chart illustrating a method whereby a stream in one stream forks a thread in another stream in a processor containing a special register transfer.

Fig. 4 is an architecture diagram illustrating interrupt mapping and processing in an embodiment of the present invention.

Description of the Preferred Embodiments

Multiple active streams operating in the same processor are often related by the nature of the threads supported. Advantages may be gained, therefore, if a thread running in one stream (an active stream) is enabled to initiate and/or control functions of one or more other active streams. Active streams may share work on the same task and may therefore need efficient methods of passing data. One active stream may temporarily require exclusive use of certain processing resources or of total throughput. Such an active stream needs a way of asserting its particular claims, while allowing other active streams to continue operating as efficiently as possible with fewer resources. These are issues in all multi-streaming processors. In this concept and the descriptions that follow, it is well to remember again that by an *active* stream is a stream that is running a particular thread, and also that a thread context is associated with an active stream by a register file.

Multi-streaming processors, as described in priority document S/N 09/216,017, now U.S. Patent 6,477,562 on 08/05/05, have physical stream resources for concurrently executing two

slave streams. If there is more than one Master stream running, each may have different designated slave streams. With appropriate control settings, active streams may act as supervisors of other active streams, temporarily (typically) controlling their execution and communicating with them. Further, a Master Stream has, and supervisor streams may have, control over what processing resources active slave streams may use, either directly or by modifying a stream's priorities.

Fig. 1A is a generalized diagram of a multi-streaming processor according to an embodiment of the present invention, showing an instruction cache 101 providing instructions from multiple threads to four streams 103, labeled 0-3, from which an instruction scheduler dispatches instructions from active streams to functional resources 107. A set of multiple register files 109, in this case four, but may be more, is shown for use in processing, such as for storing thread contexts to be associated with active streams during processing. Data flows to and from register files and a data cache 111, and the functional resources may include a Register Transfer Unit (RTU) as taught in priority document S/N 09/240,012, now U.S. Patent 6,242,887, incorporated herein by reference.

In this embodiment a unique inter-stream control bit-map 115 stores individual bits, and in some cases binary values of bit combinations, associated with individual streams and assigned particular meaning relative to inter-stream communication and control, as introduced above. A shared system bus 113 connects the instruction and data caches. The diagram shown is exemplary and general, and the skilled artisan will recognize there are a number of variations which may be made. The importance for the present purpose is in the multiplicity of streams adapted to support a multiplicity of threads simultaneously.

done, the supervising thread starts the new thread in stream 2 in step 207. Alternatively, stream 2 may be put in sleep mode, waiting on an internal or external event. The new thread starts running in stream 2 in step 208. In steps 209 and 210 both streams run independently and concurrently until a join is required. In this example, it is assumed that the thread running in stream 1 finishes first.

When the supervisor thread needs to join the forked thread, it checks first to see if the forked thread is still running. If so, it executes an instruction at step 211 that puts itself to sleep, setting the sleep bit in stream control bits 118, and then waits for a join software interrupt from the forked thread. The forked thread sends a join interrupt in step 212 and the supervisor thread receives the interrupt and wakes in step 213. The supervisor completes the join operation in step 214. Finally the forked thread ends in step 215, freeing its stream for use by another thread.

Fig. 2B illustrates the companion case wherein the forked stream finishes before the supervisor stream. In this case, when the forked stream finishes, it immediately sends the join interrupt (step 216). The interrupt remains on hold until the supervisor stream finishes, then the interrupt is serviced in step 217 and the join is completed. If registers can be loaded and stored in the background as described in co-pending priority application filed January 27, 1999, entitled "Register Transfer Unit for Electronic Processor," then the process of forking a new thread for which the context is not already loaded is modified from the steps shown in Fig. 2 as shown in Fig. 3.

As shown in Fig. 3, the steps are identical to those in Figs. 2A and 2B, except step 204 for setting program counter and context is eliminated. After step 206, in new step 301, the supervisor signals the register transfer

streams to start a thread in each of the four streams to process arriving data packets.

As is known in the art of data routers not all data packets need equal processing. Some packets need only be forwarded as received. Others may need to be restructured into a different format. Still others will need to be, for example encrypted/decrypted. The type of packet dictating the work flow to process the packet is typically contained in a header for the packet, and the type and scope for processing can only be known to the processor after a thread context is loaded to a register file, the register file is associated with a stream (active stream) and processing is commenced on a data packet.

In a preferred embodiment of the present invention, as was illustrated in Fig. 1D and described above, each stream is said to have an *execution* priority, meaning that only a process with higher priority may run in that stream. In one aspect the execution priority for each stream of a processor is maintained as three editable bits in a portion of bit map 115 of Fig. 1A. In the exemplary data router case, as packets arrive to be processed, the context for the packet is loaded to a register file. This may be done in preferred embodiments by a Register Transfer Unit (RTU) according to the teachings of priority document S/N 09/240,012, now U.S. Patent 6,292,888

As described immediately above, it is necessary to commence processing of a data packet before the nature of the packet may be known. Therefore, as packets arrive and register files are loaded, each context is given an initial high priority. For example, on a scale of seven, each initial context will be assigned a priority of six.

Now, as streams become available, register files are associated with streams, according to priority of the register files and the execution priority of the streams. Associating a register file with a stream starts the context

slave to suspend such control during critical periods using the supervisory control bit.

The types of control that one stream may have over other streams through the mechanisms of supervisory control bits and stream control bits are not limited. A single type of control or a large number of controls may be appropriate, depending on the purpose of the system. Additional controls could regulate the sharing of global registers or global memory, memory protection, interrupt priorities, access to interrupt masks or access to a map between interrupts or exceptions and streams, for example. In a processor with one or more low power modes, access to power control may also be regulated through additional supervisory control bits and stream control bits or such control may be reserved exclusively for a stream that is running the master thread.

The type of control that one stream may have over another stream's resources can also take many forms. In one of the simplest forms, a simple two-stream controller, for example, to be used in a dedicated application, with a fixed master/supervisor and a fixed slave stream, a single stream control bit for the slave stream could give the master stream the ability to disable the slave during instruction sequences when the master needs full use of all resources.

Priorities and scheduling of any form described in priority document S/N 09/216,017, now U.S. Patent 6,477,562, may be implemented in combination with the new teachings of the present invention. If such priorities are not implemented, then a stream could exert a simpler form of control by directly blocking another stream's access to one or more resources temporarily. In this case the supervisory control bits representing priorities would be replaced with bits representing resource control. Priority maps would be replaced with one or more control bits used to temporarily deny access to one or more